5th International Conference on Experiments/Process/System Modeling/Simulation/Optimization
5th IC-EpsMsO
Athens, 3-6 July, 2013

SELF-ORGANIZING MAP BASED ADAPTIVE SAMPLING

Keiichi Ito[1] *, Tom Dhaene[1], Naji El Masri*, Roberto d'Ippolito*, and Joost Van de Peer*

[1] University of Ghent, Department of Information Technology (INTEC)
Gaston Crommenlaan 8 Bus 201, 9050 Ledeberg – Gent, Belgium
e-mail: keiichi.ito@intec.ugent.be, tom.dhaene@intec.ugent.be

*Noesis Solutions
Gaston Geenslaan, 11 B4, 3001 Leuven, Belgium
e-mail: keiichi.ito@noesissolutions.com, naji.elmasri@noesissolutions.com,
roberto.dippolito@noesissolutions.com, joost.vandepeer@noesissolutions.com

Keywords: Uncertainty, Simulated-Annealing, Optimization, Self-Organizing Map, Non-Parametric Distribution, Density Learning

Abstract. *We propose a new adaptive sampling method that uses Self-Organizing Maps (SOM). In SOM, densely sampled regions in the input space is represented by a larger area on the map than that of sparsely sampled regions. We use this property to progressively tune-in on the interesting region of the design space. The method does not rely on parameterized distribution, and can sample from multi-modal and non-convex distributions. In this paper, we minimize several mathematical test functions. We also show its performance in inequality-constrained objective satisfaction problem, in which the objective is to seek diversity in solutions satisfying certain upper-bound constraint in the minimized objective. A new merit function and a measure of space-filling quality were proposed for this purpose.*

1 INTRODUCTION

In today's engineering endeavor, it is common to run computer simulations to understand the behavior of complex systems and optimize their parameters to obtain satisfactory designs before actual physical prototypes are built. The objective of the engineer is not only the optimization of the systems but also to understand what makes a good design. The question - particularly in the early stage of the design process - is often not about finding the best parameter values, but is about what parameter ranges would generate competitive designs or solutions. In a more pragmatic simulation level, engineers often want to confine the simulation runs to parameter settings for which results are trustworthy. Such information may not be available until one actually runs the simulation (e.g. whether it crashes/converges or not). Our research is motivated from engineering design situations in which accuracy of optimized result is not of paramount importance, but efficient identification of the "good input space" is.

In this paper, we propose an adaptive sampling algorithm that uses a Self-Organizing Map (SOM) [2] to perform a kind of importance sampling. SOM represents a set of multidimensional vectors and arrays in a low dimensional space - typically 2D. The map consists of cells (neurons) and each of them has an associated weight vector. These vectors are trained with a set of training samples so that average distances between the training samples and the weight vectors are minimized. Two adjacent cells indicate that two associated vectors are similar, i.e. their distance (usually Euclidean) is small. In this adaptive sampling method, the cell weight-vectors correspond to the potential sampling points. The fundamental idea is to have an algorithm that learns to sample in the region of interest in the design (or input) space using the density learning mechanism in SOM. It is similar to Monte Carlo Optimization methods such as Probability Collectives [6] and Cross-Entropy [3] Methods. However, we do not use parameterized probability density functions to represent solutions. Instead, SOM is used to obtain a set of solutions as represented by the weight vectors in each cell of the map. Since SOM represents a densely sampled region as having larger area on the map than that of a sparsely sampled region, a weight vector can be considered as being analogous to an instance of random vector drawn from a probability density distribution. Furthermore, these vectors are "mutated" to improve diversity. The training sample set can be iteratively enriched with new samples that exhibit desirable responses (or objective values) and SOM retrained on them. At the same time, the training set has a maximum capacity. The less interesting samples can be discarded from the set. This leads to SOM eventually showing only good solutions.

In the literature, application of Self-Organizing Map (SOM) to optimization problems can be found in [8, 1, 5]. Su and Zhao [8] proposed SOM-based optimization (SOMO), in which the winning weight vector at each iteration is the weight vector that gives maximum (or minimum) objective function value. Neighboring weight vectors are updated to be closer in Euclidean distance to this winning weight vector. For this method there is no sample set to train the weights. It just needs initialization of the weight vectors. A small perturbation is added to every update of the vectors to keep diversity.

Chen and Young's [1] SOM-based search algorithm (SOMS) also updates the weight vectors according to their corresponding objective values. In their method, the training objective is to align distances represented as Gaussian distribution functions in the map (or neuron) with the weight vector (or parameter) space by iteratively adjusting its center to the weight vector returning the minimum (or maximum) objective value found so far and its standard deviation to the distance between the center of current iteration and the newly found best point. Other off-center weight vectors are updated accordingly so that Gaussian distance from center on the map and the Gaussian distance from the center in the weight vector space is minimized using a gradient based algorithm. Therefore, the map (or neuron space) gives the reference distribution of samples and this distribution is mapped in the input space with new center and standard deviation in the input space. The center of the map, standard deviation and the weights are updated per dimension. Thus, if the map is rectangular and the input space is two dimensional, the weight vectors are distributed in rectangular shape changing its aspect ratio and area as the iteration proceeds.

Milano, Koumoutsakos, and Schmidhuber [5] proposed Evolutionary Strategy (ES) based methods in which mutation is performed using SOM, namely Kohonen SOM ES (KSOM-ES) and Neural-Gas SOM ES (NGSOM-ES). In KSOM-ES, the weight vectors partition the search space into simplexes and at each iteration a simplex is randomly chosen to sample a point uniformly. In NGSOM-ES, a sample is taken from normal distribution around one of the SOM weight vectors. If the sample gives better objective value than what is known, the weight vector updates take place.

In all of these, it is the training algorithm of the weight vectors that is modified so as to suit to the task of optimization. They do not train from the training sample distribution but from the point giving the best objective function value. They also assume the optimal point is unique. In these algorithms, there are no mechanism to track multiple interesting points or to identify a solution "set" or input region. Our algorithm does not entail any modification of existing SOM. Therefore, different implementation of SOM or other density learning algorithms can be used in its place. It provides a convenient functionality to algorithmically focus on a more interesting region of the input space by shifting the sample densities.

2 ALGORITHMS

Algorithm 1 shows a high level description of our Self-Organizing Map Based Adaptive Sampling (SOMBAS). In every iteration, the trained Self-Organizing Map (SOM) produces new solution candidates. Weight vectors are selected based on its objective or merit value according to a method similar to Simulated Annealing. Perturbations are applied to these selected vectors and their objective values are computed. These selected samples are included in the training sample set to train the SOM of next iteration.

Algorithm 1 SOM Based Adaptive Sampling

```
1: Sample randomly (or according to DOE) N samples to
   create initial training set
2: while Termination condition not met do
3:     Normalize the training samples
4:     Train SOM using the normalized training set
5:     for all cells satisfying SELECTION CONDITION
       do
6:         Perturb the selected weight vectors (cell weight
           vectors) according to MUTATION
7:     end for
8:     Evaluate true output of the perturbed samples
9:     Un-normalize the perturbed samples
10:    UPDATE TRAINING SET
11: end while
```

Algorithm 2 SELECTION CONDITION

1: Let $y_{\min}$ be smallest output in the training sample set X, y be output from a cell weight vector, and T be the selectivity parameter (or Temperature)
2: Generate a uniform random number $0 \leq r < 1$ and check the following:
3: **if** $r < \exp\left(\frac{y_{\min} - y}{T}\right)$ **then**
4: Corresponding weight vector is selected
5: **end if**

The selection condition as described in Algorithm 2 took hint in the Simulated Annealing literature. The probability of a weight vector being selected depends on how close it is from the weight vector with the smallest estimated objective value. All the vectors will have estimate of objective values because the training samples are concatenated vector of input and output (x^T , y). The selection condition is

$$r < \exp\left(\frac{y_{\min} - y}{T}\right), \tag{1}$$

where r is a random number drawn from a uniform distribution and is $0 \leq r < 1$, ymin be smallest output in the training sample. The temperature T defines how selective the condition is and smaller value gives smaller number of new samples to be added to the training data set. For our algorithm, $0.01 \leq T < 10$. Unlike SA, T is held constant. For design space identification, we consider a case in which we seek to minimize an objective value y below certain threshold L . One idea is to use a merit function similar to those described by Torczon et al [9]. One could give better chance of being selected to points (i.e. cell weight vector) that are distant from existing training samples regardless of y's value. To achieve this, we propose the following formula for the merit function.

$$F = \max(L, y) - \rho \min(\|\boldsymbol{s} - \boldsymbol{t}_i\|_2), \qquad i = 1,2, \dots, N, \tag{2}$$

where $\boldsymbol{s}$ is the input vector for which F needs to be minimized, $\boldsymbol{t}_i$ is a set of target samples from which minimum distance to the input vector $\boldsymbol{s}$ is calculated, N is the number of such target vectors, and ρ is a weight constant. To minimize this merit function, one needs $y < L$ and maximize the distance to the nearest target vector $\min(\|\boldsymbol{s} - \boldsymbol{t}_i\|)$. In our case, target vectors are the training set and the input vector $\boldsymbol{s}$ is the selected weight vector from SOM. The algorithm to replace the output with this merit function is described in Algorithm 3. If y is greater than the threshold L, both y and the new weight vector's distance from the training set are taken into account. If y is less than L, then the distance to the nearest training vector is the only term affecting the objective value and smaller F is obtained when the weight vector's distance to the nearest neighbor is larger. The ρ in equation 2 is a positive weighing constant.

Algorithm 3 MERIT FUNCTION

1: Let Trunc denote the value below which objective or output y is considered to be "good enough", $\mathbf{s}$ denote a weight vector $(\mathbf{x}^T, y)$ from SOM, and $\mathbf{t}_{i=1,2,\dots,N}$ denote the training samples
2: **if** Trunc is specified **then**
3: Normalize Trunc ($\mathbf{s}$, and $\mathbf{t}_i$ are already normalized)
4: $y \leftarrow \max(\text{Trunc}, y) - \rho \min(\|\mathbf{s} - \mathbf{t}_i\|_2)$
5: Normalize y
6: **end if**

Mutation as described in Algorithm 4 is applied to the selected weight vectors. We use the vectors as the centers of multi-variate Gaussian distributions. The covariance matrix is obtained from the selected weight vectors. we use an idea from CMA-ES to the updating of covariance matrices. Covariance in current iteration is combined with the covariance computed in the previous iteration: $0.2C + 0.8C_{old}$. This is to avoid adapting too quickly to local minima. On top of that, we multiply a factor which is different whether the previous iteration produced a new minimum or not. If the previous iteration achieved a new minimum we apply an expansion factor F_e, which we assign a real value larger than 1. On the other hand, if the previous iteration did not produce

a smaller minimum compared to that of two iterations ago, we multiply a contraction factor F_c, which we assign a real value between 0 and 1. Covariance matrix is the same for all the selected weight vectors. Each weight vector is perturbed by sampling from their Gaussian distribution. This can easily be done using numpy module in Python. Mutation is very important to avoid premature convergence in SOMBAS.

Algorithm 4 MUTATION

Let contraction factor be F_c and expansion factor be F_e
Let mutation probability be P_m
Given training samples, compute covariance matrix C, and let the covariance matrix from previous iteration be C_{old}
if current $y_{\min}$ < previous $y_{\min}$ **then**
    $C = F_e\ (0.2C + 0.8C_{\text{old}})$
else
    $C = F_c\ (0.2C + 0.8C_{\text{old}})$
end if
For each selected sample, replace it by sampling from multivariate normal distribution with center at the selected sample with covariance C.

After the perturbation, the new set of samples (weight vectors) is supplied to the training set and the training set needs to be updated. Algorithm 5 and Algorithm 6 are two such methods. Algorithm 5 has a faster convergence but is more prone to loose diversity in training set compared to Algorithm 6. In the next section we will use Algorithm 6.

Algorithm 5 UPDATE TRAINING SET 1

Add the perturbed samples to the training set
if Training set sample size > maximum sample size **then**
    Sort the training set with respect to output value
    Remove the worst samples to make the training sample size equal to maximum sample size
end if

Algorithm 6 UPDATE TRAINING SET 2

for all perturbed weight vectors' response y_p **do**
    Randomly pick one of the training sample, and obtain its response y_t
    if $y_p < y_t$ **then**
        Replace the training sample with the perturbed weight vector
    end if
end for

3 EXPERIMENTS

Two kind of experiments were tried out to see the characteristics of the algorithm. One is optimization and the other is design space identification. Unlike classification problems in which the training samples contain both positive and negative training samples, design space (feasible domain) identification can start from 100 % negative (infeasible) training samples. Therefore, the search mechanism (optimization) and space-filling characteristics are both important. In the first kind, the objective is to find the input parameters for which the objective value is minimum and the second kind is to obtain as diverse set of input parameters as possible satisfying the objective value constraint. For these experiments some well known test functions are used. We employed Differential Evolution's result for comparison. Python script of DE was implemented. DE was selected for comparison because it also uses distribution of samples (population) during the course of evolution. The objective in this section is to illustrate the different characteristics between DE and our SOM Based Adaptive Sampling (SOMBAS). In all of the experiments conducted, we used the self organizing map script from Marsland's book [4]. It was treated as black box and its learning parameters were kept to the default value in

the script, except for the map size (number of cells in x,y directions) and number of training iterations.

3.1 Function Minimization

To investigate optimization functionality of SOMBAS,We tried 5 functions: Rosenbrock, Rasgtrigin, Rotated Ellipsoid, Ackley, and Manevich. The result of optimizations are compared against DE. We employed the classical DE1 as described in Storn and Price[7]. The stopping criteria for these functions were the best solution reaching the objective value below 1.0×10^{-6}, or maximum number of function evaluation is reached, or the difference between the worst sample and best sample in the training set in SOMBAS or population in DE becomes less than 1.0×10^{-6}. We refer this difference as "gap tolerance". We measured number of function evaluation $\boldsymbol{N_f}$ and the objective values $\boldsymbol{f}$. Optimization were run multiple times and average value of number of function evaluation $\tilde{\boldsymbol{N}}_{\boldsymbol{f}}$ and average minimum objective value reached $\tilde{\boldsymbol{f}}$ were computed. The parameters of SOMBAS and DE were manually tuned, but to the extent that they solve the problem without excessively premature convergence or excessively large number of iterations. Therefore, the comparative results shown in this section does not indicate any definite superiority or inferiority against the algorithm compared. The global minima of the 5 functions are 0 and the corresponding input values are $\boldsymbol{x_i = 1}$, $\boldsymbol{i = 1, \dots, D}$ for Rosenbrock and $\boldsymbol{x_i = 0}$ for the remaining four functions. Here, D is the number of dimensions.

The results in two dimensional problems are shown in Table 1. They show the average of 20 runs. SOMBAS showed a particularly strong performance in an epistatic problem (Rosenbrock) and in an ill-conditioned problem (Manevich). Rastrigin and Ackley are highly multimodal function but non-epsitatic, and in these, SOMBAS performed less well.

	SOMBAS		*DE*	
Function	$\tilde{N}_f$	$\tilde{f}$	$\tilde{N}_f$	$\tilde{f}$
Rosenbrock	1248	$3.43e-07$	2137	$3.75e-07$
Rastrigin	2218	$4.26e-07$	2076	$4.28e-07$
Rotated Ellipsoid	936	$3.77e-07$	1196	$4.22e-07$
Ackley	1065	$5.19e-07$	1545	$6.72e-07$
Manevich	514	$4.16e-07$	1050	$4.79e-07$

Table 1. Optimization results of 2D functions (average of 20 runs)

In 30 dimensional problems, number of training samples in SOMBAS was matched to the population sizes of DE and they were between 20 and 45 depending on the function to be minimized. These population sizes are known to be optimal for DE[7]. The number of SOM weight vectors were set slightly larger than the number of training samples. Number of function evaluations and objective values were listed at around 2,000, 20,000 and 200,000 function evaluations in Table 2 Table 3 and Table 4 respectively. For Table 2 and Table 3, the numbers are average of 20 runs and for Table 4 they are the average of 5 runs.

	SOMBAS		*DE*	
Function	$\tilde{N}_f$	$\tilde{f}$	$\tilde{N}_f$	$\tilde{f}$
Rosenbrock	2019	193	2025	$4.25e+05$
Rastrigin	2013	189	2030	293
Rotated Ellipsoid	2003	63.5	2010	418
Ackley	2005	4.08	2020	6.12
Manevich	2014	0.0177	2010	0.101

Table 2. Optimization results of 30D functions after 2,000 function evaluations (average of 20 runs)

In Table 3, we observe that Ackley and Manevich function converges prematurely for SOMBAS and Manevich converges successfully in about 13,000 function evaluations for DE. Smaller gap tolerance (and larger number of SOM weight vectors) may avoid the premature convergences, but we will not delve further on this as it is not our objective.

	SOMBAS		*DE*	
Function	$\tilde{N}_f$	$\tilde{f}$	$\tilde{N}_f$	$\tilde{f}$
Rosenbrock	20018	59.4	20025	683
Rastrigin	20014	48.1	20020	76.8
Rotated Ellipsoid	20012	5.55	20010	18.3
Ackley	17691	1.48	20020	0.000373
Manevich	14310	$1.46e-06$	13163	$9.41e-07$

Table 3. Optimization results of 30D functions after 20,000 function evaluations (average of 20 runs)

Table 4 shows that by 200,000 function evaluations DE has minimized the function values below the 1.0×10^{-6} threshold except for Rosenbrock function. For SOMBAS, Rastrigin converged to a local optima and Rotated Ellipsoid has not converged at 200,011 evaluations. DE seems to have an edge in the final accuracy of the optimization in this table. One particular aspect observed in these tables showing results at three different number of function evaluations is that SOMBAS perform rather fast decrease in function values at early stages of the optimization runs. To see this more clearly, we performed an additional optimization campaign with larger number of training samples or population.

	SOMBAS		*DE*	
Function	$\tilde{N}_f$	$\tilde{f}$	$\tilde{N}_f$	$\tilde{f}$
Rosenbrock	190155	3.12	200025	7.21
Rastrigin	63723	18.7	115031	$8.97e-07$
Rotated Ellipsoid	200011	0.000399	152358	$9.65e-07$
Ackley	*n.a.*	*n.a.*	31104	$9.60e-07$
Manevich	*n.a.*	*n.a.*	*n.a.*	*n.a.*

Table 4. Optimization results of 30D functions after 200,000 function evaluations (average of 5 runs)

Table 5 shows the results with training sample and population size of 900 and total function evaluation of about 2000. In this table, the minimum function values reached by SOMBAS is substantially lower than those reached by DE. Also, comparing to the function values attained in Table 2, DE has shown a lot larger increases in function values compared to the increases for SOMBAS. The difference between Table 2 and Table 5 is the number of samples or population evaluated in each iteration, that of Table 5 being much larger. This indicates that for a fixed number of function evaluations, the minimum function values achieved is less sensitive to the increase in number of training samples in SOMBAS than to the increase of population in DE.

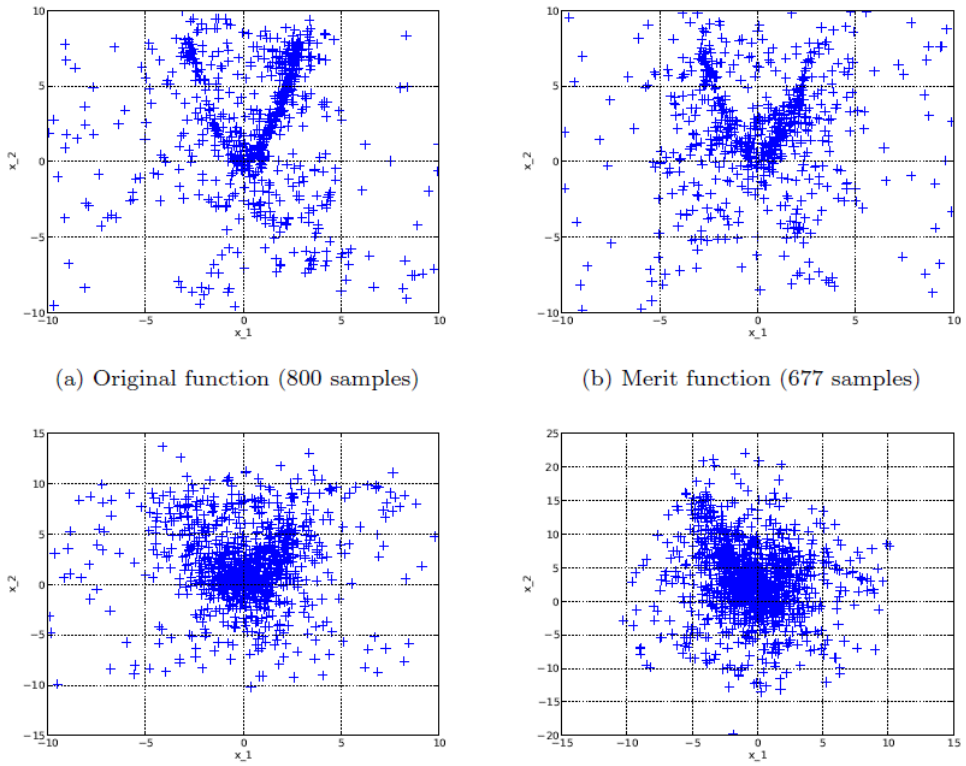
3.2 Design Space Identification

In this subsection, we consider a kind of constraint satisfaction problem in which there is a constraint on the objective (to be below certain threshold value) but one would like to know what kind of inputs to the objective function would satisfy this condition. Ideally, one would like to have as much variety as possible in the collection of inputs that we obtain as solutions.

Figure 1 shows all the sampled points by SOMBAS in blue crosses during the course of satisfying maximum objective value returned from the training samples to be less than 100 on Rosenbrock function. This figure demonstrates the effect of having the Merit Function and/or Mutation instead of simply using the objective function in the space filling task. In general, Merit function is useful in decreasing large sampling gaps as shown in Figure 1 (a) and (b). The number function evaluations until all samples returns objective value below the threshold) Trunc = 100 in Algorithm 3 do not increase by its use. On the other hand, Mutation carries some penalties in number of function evaluations. However, it has additional space-filling characteristics not attainable by use of only the Merit Function.

	SOMBAS		*DE*	
Function	$\tilde{N}_f$	$\tilde{f}$	$\tilde{N}_f$	$\tilde{f}$
Rosenbrock	2283	201	2700	$1.33e+06$
Rastrigin	2083	219	2700	731
Rotated Ellipsoid	2379	11.9	2700	533
Ackley	2353	2.38	2700	12.5
Manevich	2196	0.0982	2700	3.41

Table 5. Effect of large training samples or populations (900) in optimization of 30D functions for relatively small number of function evaluations (2000) (average of 20 runs)



(a) Original function (800 samples) (b) Merit function (677 samples)

(c) Mutation and original function (1009 samples) (d) Mutation and merit function (1285 samples)

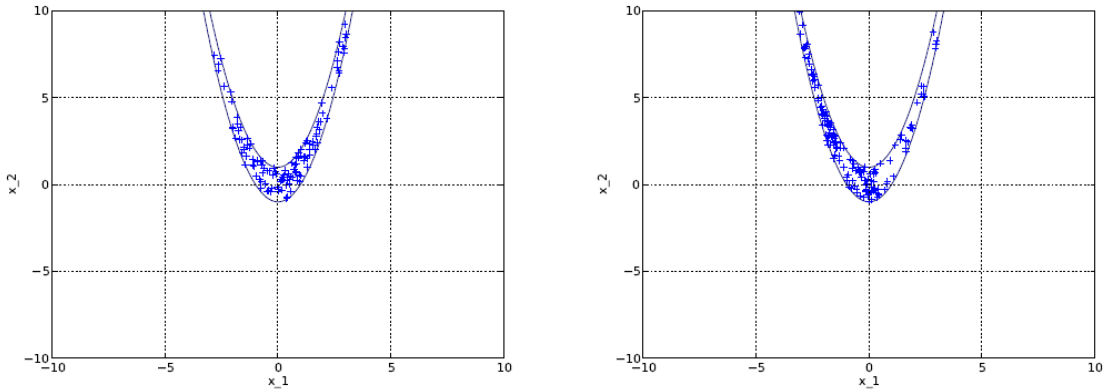
Figure 1. Sampled sites in the process of satisfying the condition that maximum response of the training samples be less than 100

To see the effect of Merit Function and selectivity temperature T, we run 20 times the constraint satisfaction problem and summarized in Table 6. It is again Rosenbrock function in 2D and all the training samples (60 of them) were driven to be less than 100. In this table, we see four rows with different setting for selectivity temperature T and Merit Function. In all four cases, Mutation was applied. If "Trunc" is none, the Rosenbrock function output is used in the selection of weight vectors. If Trunc is 100, Merit Function is used (L = 100 in Equation 2). The σs indicate the standard deviation of x_1, x_2, and function value f in the final training samples. The larger these σs are the more diverse is the solution set. Here, we see that Merit Function indeed increases the standard deviation of the solution set (or the final training samples).

Trunc	T	$\tilde{N}_f$	$\tilde{f}$	σ_{x_1}	σ_{x_2}	σ_f
None	0.1	306	0.531	1.07	1.74	17.5
None	1.0	564	0.301	1.38	2.51	13.3
100	0.1	310	0.616	1.47	2.49	27.3
100	1.0	542	0.443	1.45	2.47	16.6

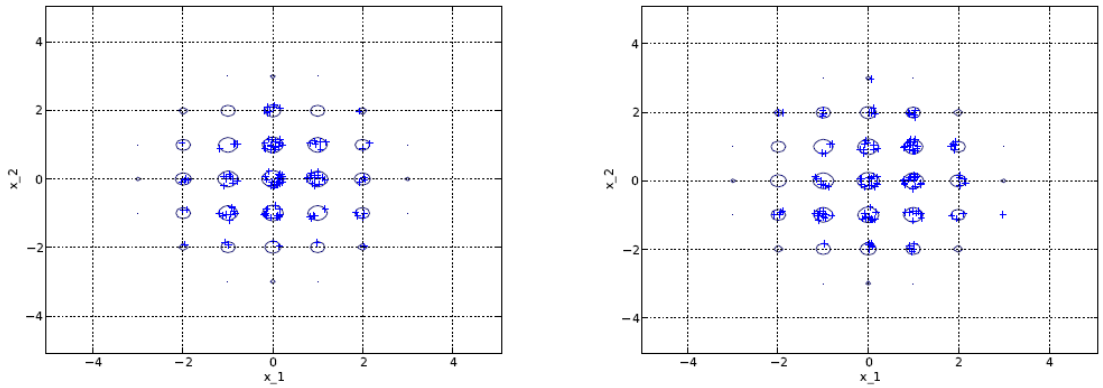
Table 6. Effect of Truncation and Selectivity Parameter "T" on Diversity in Solutions Satisfying Objective Constraint (average of 20 runs)

In Figure 2 through Figure 4, we visualized different types of feasible domains and the sampling (shown in cross) inside them. The contour plot show the boundaries of the domains. Except for Rastrigin function, SOMBAS was able to produce a fairly uniform space-filling of samples in the 2D input domain for the functions tested. Since DE also have distribution learning characteristics, did very well in the feasible domain space-filling task.



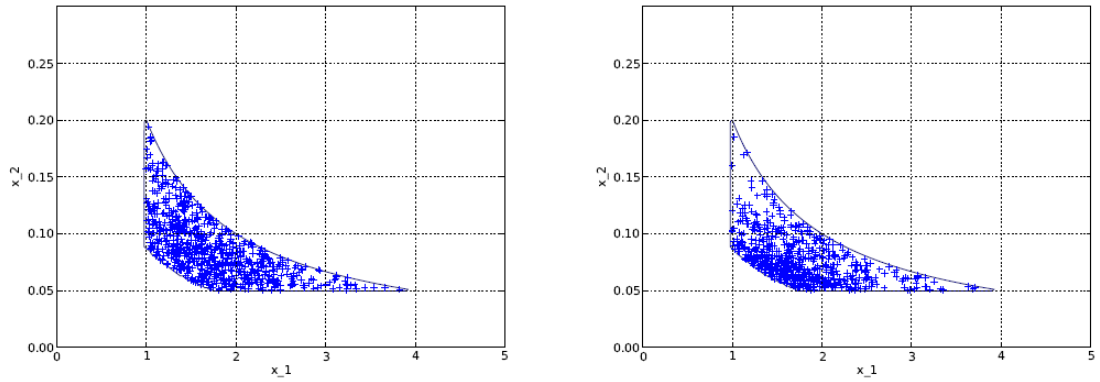
(a) SOMBAS (b) Differential Evolution

Figure 2. Sample distribution satisfying objective condition $f \leq 100$ in Rosenbrock function



(a) SOMBAS (b) Differential Evolution

Figure 3. Sample distribution satisfying objective condition $f \leq 10$ in Rastrigin function



(a) SOMBAS (b) Differential Evolution

Figure 4. Sample distribution satisfying constraints in Hollow Beam function

To see this in a more statistical ground, we repeated the sampling task for each of the three equation 20 times. Table 7 to Table 9 show the results. Here, $\tilde{\tilde{\boldsymbol{d}}}$ is the average distance of nearest neighbors. The tilde on top of the symbol signifies averaging. The nearest neighbor have two tildes corresponding to the average in the feasible domain and average of 20 runs. $\tilde{\boldsymbol{N}}_{\boldsymbol{f}}$ is the average number of function evaluation, $\tilde{\boldsymbol{N}}_{\boldsymbol{s}}$ is the average number of samples in the feasible domain, $\tilde{\boldsymbol{\sigma}}_{\boldsymbol{d}}$ is the average standard deviation of distances to the nearest neighbors. We also define the feasible rate $\frac{\tilde{\boldsymbol{N}}_{\boldsymbol{s}}}{\tilde{\boldsymbol{N}}_{\boldsymbol{f}}}$ and coverage length $\tilde{\boldsymbol{l}} = \tilde{\tilde{\boldsymbol{d}}} \times \tilde{\boldsymbol{N}}_{\boldsymbol{s}}$. The feasible rate gives the ratio of number of feasible samples to the total number of function evaluation. Larger the value the better. The coverage length gives efficiency of space-filling quality. The larger its value the better. The coverage length is meaningful only when we compare different methods on the same feasible domain identification problem. We need further investigation in the measurement of space-filling quality that is scale independent.

Algorithm	$\tilde{\tilde{d}}$	$\tilde{\sigma}_d$	$\tilde{N}_s$	$\tilde{N}_f$	$\frac{\tilde{N}_s}{\tilde{N}_f}$	$\tilde{l} = \tilde{\tilde{d}} \times \tilde{N}_s$
SOMBAS	1.7	1.4	129	656	0.20	213
DE	1.7	1.3	114	621	0.18	188

Table 7. Average Nearest Neighbor Distances of Sampled Points by SOMBAS and DE - Rosenbrock Function

Algorithm	$\tilde{\tilde{d}}$	$\tilde{\sigma}_d$	$\tilde{N}_s$	$\tilde{N}_f$	$\frac{\tilde{N}_s}{\tilde{N}_f}$	$\tilde{l} = \tilde{\tilde{d}} \times \tilde{N}_s$
SOMBAS	0.53	0.33	120	1501	0.0801	64
DE	0.56	0.34	112	1005	0.112	63

Table 8. Average Nearest Neighbor Distances of Sampled Points by SOMBAS and DE - Rastrigin Function

Algorithm	$\tilde{\tilde{d}}$	$\tilde{\sigma}_d$	$\tilde{N}_s$	$\tilde{N}_f$	$\frac{\tilde{N}_s}{\tilde{N}_f}$	$\tilde{l} = \tilde{\tilde{d}} \times \tilde{N}_s$
SOMBAS	0.15	0.12	171	377	0.454	26
DE	0.18	0.12	137	469	0.290	24

Table 9. Average Nearest Neighbor Distances of Sampled Points by SOMBAS and DE - Hollow Beam

In all three functions tested, we matched the training sample size in SOMBAS with the population size in DE. This was done so that space-filling density becomes more or less comparable between DE and SOMBAS. SOMBAS showed very good feasible rates and coverage lengths compared to DE for Hollow Beam and Rosenbrock. On the other hand, DE outperformed SOMBAS in Rastrigin Function in both feasible rate and coverage length.

4 CONCLUSIONS

Although, the conducted experiments are on a series of simple mathematical functions, they have helped in elucidating the strength and weakness of the new algorithm. In particular, it has shown promises in efficient initial reduction in function values in optimization as well as economical identification of input spaces that satisfy objective criteria.

Future work will include application of the algorithm to industrial problems as well as to surrogate model construction. A good measure of space-filling quality needs to be investigated. Since Self-Organizing Maps can visually represent high dimensional space in 2D, it can provide a way to include human in the optimization and adaptive sampling loop. This feature may be particularly useful when problem definition is not rigorous and human experience plays a substantial role in the design process. We would also be interested in further investigating the parallel capability and efficiency of the algorithm.

5 ACKNOWLEDGMENTS

Authors would like to thank IWT for the funding of this research through the Baekeland Mandate program.

5 REFERENCES

[1] Chen ,Y.-Y. and Young, K.-Y. (2007), An SOM-based algorithm for optimization with dynamic weight

updating. International Journal of Neural Systems, 17:171 – 181.

[2] Kangas, J. and Kohonen, T. (1996), Developments and applications of the self-organizing map and related algorithms. Mathematics and Computers in Simulation, 41:3 – 12.

[3] Kroese, D. P., . Porotsky, S, and Rubinstein, R. Y. (2006), The Cross-Entropy method for continuous multi-extremal optimization. Methodology and Computing in Applied Probability, 8:383–407.

[4] Marsland, S. (2009), Machine Learning: An Algorithmic Perspective. CRC Press, New Jersey, USA.

[5] Milano, M., Koumoutsakos, P., and Schmidhuber, J. (2004), Self-organizing nets for optimization. IEEE Transactions on Neural Networks, 15(3):758–765, May.

[6] Rajnarayan, D., Wolpert, D., and Kroo, I. (2006) Optimization under uncertainty using probability collectives. In 11th AIAA/ISSMO Multidisciplinary Analysis and Optimisation Conference, Portsmouth, Virginia, 6 – 8 September.

[7] Storn, R. and Price, K. (1995), Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704, USA.

[8] Su, M.-C. and Zhao, Y. X. (2009), A variant of the SOM algorithm and its interpretation in the viewpoint of social influence and learning. Neural Computing and Applications, 18:1043 – 1055.

[9] Torczon, V. and Trosset, M. W. (1998), Using approximations to accelerate engineering design optimization. Technical report, Institute for Computer Applications in Science and Engineering (ICASE).